Replit: Build, Run, Share and Deploy an AI App

Replit is a browser-based workspace where you write code, run it, and publish it on the web, with no installation on your computer. This guide goes from your first running page to an AI-powered app with secure keys, saved data, and a deployed address. Plans and prices are deliberately not covered here; check Replit's own pricing page when you need them.

> **Running example.** A small "Dish Description Studio": a form takes a dish name, category and price, calls an AI model, and shows a short description. It is the same food-seller example used in the tokens guide.

Part A. Basics

1. Words you need

Word	Meaning
Repl	A project: its files, its running app and its settings.
Workspace	The browser editor: file tree, code editor, console and preview.
Console	Where the program prints messages and errors.
Preview	The running app shown inside the workspace.
Secret	A private value (such as an API key) stored outside the code.
Deployment	A published copy of the app at a public address.
Replit Agent	An AI that builds or edits the project from your written request.
Environment variable	A named setting the program reads at runtime, such as `AI_API_KEY`.

2. Create an account and your first project

1. Sign up at replit.com. For practice, use a test account, not your work or personal main account.
2. Click **Create Repl**. Choose a template for a web app in a language you know (for example Python with a web framework, or Node.js).
3. Give the project a clear name, such as `dish-studio-practice`.

4. Press **Run**.

You should see: the console shows the server starting, and the preview pane shows a page. If the preview is blank, look at the console first.

3. Learn the workspace in five minutes

Area	What you do there
File tree (left)	Create, rename and delete files. Keep the main file easy to find.
Editor (centre)	Write code. Save happens automatically in most setups.
Console (bottom)	Read output and errors. Read the first error line, not the last.
Preview (right)	See the running page. Refresh after a change if needed.
Packages panel	Add libraries your code imports.
Secrets panel (lock icon)	Store API keys and private values.
Version history	See earlier saved states and roll back a change.

4. Make one change and see it run

1. Open the main file and change the page heading to `Dish Description Studio`.
2. Save, then refresh the preview.
3. If nothing changes, read the console for a red error line.

Common error: a missing package. Read the error ("No module named ..." or "Cannot find module ..."), add the package in the Packages panel, and run again.

5. Build the input form

Add a form with three fields: dish name, category (a dropdown: Breakfast, Lunch, Dinner, Dessert), and price in rupees. Use a submit button. Give every field a visible label. Labels help screen-reader users and make the form easier to use with a keyboard.

```
<form method="post" action="/describe">
  <label>Dish name <input name="dish" required></label>
  <label>Category
    <select name="category">
      <option>Breakfast</option><option>Lunch</option>
      <option>Dinner</option><option>Dessert</option>
    </select>
  </label>
  <label>Price (Rs) <input name="price" type="number" min="1" required></label>
  <button type="submit">Describe my dish</button>
</form>
```

You should see: the form in the preview. Submitting it does nothing useful yet; that is the next part.

Part B. Intermediate: connect an AI model safely

6. Store the key as a secret

An AI call needs an API key from your model provider. Never put the key in the code, in a screenshot, in a chat, or in a public project.

1. Create a key in your AI provider's console, with a monthly spending limit set there.
2. Open the **Secrets** panel and add `AI_API_KEY` with the key as its value.
3. In code, read it from the environment: Python `os.environ["AI_API_KEY"]`; Node `process.env.AI_API_KEY`.

> **If a key leaks,** delete it in the provider console at once and create a new one. Changing the code does not undo an exposed key.

7. Write the AI call with limits

Every call should have three guards: a fixed instruction, a capped output length, and a logged usage count. These make cost predictable (see the tokens guide).

```
import os
# Use your provider's official SDK. Names below are placeholders.
from ai_client import Client

client = Client(api_key=os.environ["AI_API_KEY"])

SYSTEM = ("You write short, appetising menu descriptions for local food sellers. "
          "Warm tone. Maximum 50 words. No prices or claims about health.")

def describe(dish, category, price):
    user = f"Dish: {dish}\nCategory: {category}\nPrice: Rs {price}"
    reply = client.generate(system=SYSTEM, user=user, max_tokens=120,
temperature=0.7)
    print("tokens in/out:", reply.usage.input_tokens, reply.usage.output_tokens)
    return reply.text
```

You should see: after one submission, the console prints the token counts and the preview shows a description of about 50 words.

8. Handle failure gracefully

- If the key is missing, show "AI is not configured" instead of a crash.

- If the call times out, show a retry message and do not retry in a tight loop.
- If the output is empty, show a fallback message and log the input.

9. Save results to a database

To keep a history of generated descriptions, add a small database. Replit offers a built-in database option in the workspace; alternatives include a hosted database service. Store only the fields you need: dish, category, price, description, time, and the token counts. Do not store personal data for a practice app.

Table: descriptions	Type	Note
id	number	Primary key, automatic
dish, category	text	From the form
price	number	Validated: greater than zero
description	text	From the AI call
input_tokens, output_tokens	number	For cost tracking
created_at	time	Automatic

Part C. Advanced: deploy, monitor and collaborate

10. Deploy the app

1. Open the **Deploy** panel and choose the type that fits: a website or web app (always-on address) is the usual choice for this example. Scheduled or background types suit other jobs.
2. Check the build command and run command the panel shows. They should match how you ran the app.
3. Deploy, then open the public address in a private browser window.
4. Test a normal submission, an empty form, a negative price, and a simulated AI failure.

You should see: the same page as in the workspace, at a public address, with secrets working. Deployment options and limits change; confirm them on Replit's documentation before you rely on them.

11. Logs, errors and rollback

- **Logs:** check the deployment logs after each change. Log the request type, the status and the token counts, never the API key.
- **Rollback:** use version history to return to an earlier saved state if a change breaks the app.

- **Health check:** keep one page that returns "ok" so you can see at a glance that the app is up.

12. Collaboration

Invite a teammate to a project only for practice work with fake data. Agree who edits which file. Use version history and comments to explain changes. Do not share secrets with collaborators; they should add their own key for their own testing.

13. Using the Replit Agent well

1. Describe the app in one paragraph: who uses it, what the inputs are, what the outputs are, and what it must never do (for example: never store customer phone numbers).
2. Ask for one feature at a time, then test it.
3. Read every file the Agent changes. Ask it to explain anything you do not understand.
4. Never let the Agent paste a real key into code. Check the secrets panel yourself.

Part D. Other tools you may need

Tool	Why you need it	Safe practice
AI provider account (console)	Gives the API key for model calls	Set a spending limit; use a test key
GitHub	Stores code history outside Replit, and lets you move projects	Use a practice repository with no secrets in it
A database (built-in or hosted)	Stores generated results	Store only the fields you need
Spreadsheet (for example Google Sheets)	Quick data sink or log for non-technical teammates	Test sheet only
Browser and dev tools	Inspect network calls and errors	Never paste keys into screenshots
Email or messaging (optional)	Notifications to a person	Approval before sending; test account only

Part E. When Replit is the right choice

Choose Replit when	Choose another tool when
You want to build and deploy a web app from a browser	You only need a no-code workflow between apps (use n8n or Make)

You are learning to code and want instant feedback	Your app needs heavy, specialised infrastructure from day one
You need a prototype for users to test quickly	You need strict data residency rules that the platform cannot meet (check first)

Part F. Test checklist

Test	Pass when
App runs in the workspace	No red error in the console
Key comes from secrets only	Code contains no key; the call works
Output is capped	Description stops near 50 words
Missing key	A clear message, no crash
Negative or empty price	Validation message; no database write
Deployed address works	Normal path returns a page
Rollback works	An earlier version restores the page

Part G. Common problems

Problem	Fix
"Module not found"	Add the package in the Packages panel and run again.
Key error at runtime	Check the secret name matches the code exactly.
Works in the workspace, fails when deployed	Check the run command and the port setting in deployment.
Blank preview	Read the console; the first red line is the cause.
Output too long, costs rising	Lower max tokens; ask for a word limit; check usage logs.

Part H. Quick check

1. Where should an API key be stored?
2. What is the first thing to read when the preview is blank?
3. Why cap output length on every AI call?
4. What is a safe way to share a practice project?

Answers: 1. In Secrets, read from the environment. 2. The first red line in the console. 3. It keeps cost and response size predictable. 4. Share only fake data, and never share secrets.

Glossary

Term	Plain meaning
Repl	A project in Replit.
Secret	A private value kept outside the code.
Deployment	A published version at a public address.
Environment variable	A named setting the program reads at runtime.
Version history	Saved earlier states you can return to.

- Replit documentation: docs.replit.com.

Part I. Build a full app, step by step (fake data only)

Step 1: plan the app on one page

Item	Example for the Dish Description Studio
Who uses it	A test seller (fake account)
Inputs	Dish name, category, price
Outputs	A 50-word description and a saved history
Must never	Store real phone numbers, or call an AI without a cap
Done when	A test seller can submit, see the description, and find it in history

Step 2: build the form, then the route

1. Build the form from the basic guide and confirm every field shows a label.
2. Add one route that accepts the form and returns the description (a placeholder reply first).
3. **Pass when:** the placeholder appears in the preview for a valid submission.

Step 3: validate on the server

- Reject an empty dish name and a price that is zero or negative.
- Return a clear message, not a crash, for each rejection.
- **Pass when:** five bad inputs each return a readable message and nothing is saved.

Step 4: connect the AI call with limits

1. Store the key as a secret. Read it from the environment only.
2. Cap output length, set temperature, and log token counts.
3. **Pass when:** the description stops near 50 words, the log shows token counts, and no key appears in code or logs.

Step 5: save history

1. Create the table from the basic guide with only the fields you need.
2. Write one row per successful call. Do not write rows for rejected inputs.
3. **Pass when:** a history page lists saved rows, newest first.

Step 6: test the failure paths

Test	Pass when
Secret removed	"AI is not configured" message, no crash
AI call times out	Retry message; no tight retry loop
Empty model reply	Fallback message; input logged without personal data
Very long dish name	Input capped and explained

Part II. Logs, versions and rollback

- **Logs:** record the route, status, duration and token counts. Never record keys or personal data.
- **Version history:** save a named checkpoint before each risky change.
- **Rollback:** if a change breaks the app, restore the last checkpoint, then find the cause in the console.
- **Health check:** one route that returns "ok" lets you check the app is up at a glance.

Part III. Collaboration and review

1. Invite only a practice teammate, and only to a project with fake data.
2. Agree who edits which file, and write a one-line note on each change.
3. Each person adds their own test key; nobody shares a key.
4. Review every change the Agent makes before you accept it.

Part IV. The Replit Agent, used well

Do	Avoid
Describe the app, the inputs, the outputs and the limits in one paragraph	Asking for the whole app in one request
Ask for one feature, then test it	Accepting large unreviewed changes
Ask it to explain code you do not understand	Pasting a real key into the chat
Check the secrets panel yourself	Trusting that a key was stored safely

Part V. Other tools you will need, and why

Tool	Use	Safe practice
AI provider console	API key and spending limit	Set a limit; use a test key
GitHub	Backup and move projects	No secrets in the repository
Database (built-in or hosted)	Saved history	Store only needed fields
Spreadsheet	A simple log for teammates	Test sheet only
Browser developer tools	Inspect requests and errors	Never share screenshots with keys

Part VI. Full glossary and quiz

Term	Plain meaning
Repl	A project in Replit.
Secret	A private value kept outside the code.
Deployment	A published version at a public address.
Checkpoint	A saved version you can return to.
Route	A web address in your app that does one job.

1. Where should an API key be stored?
2. Why validate on the server as well as in the form?
3. What should a rejected input write to the database?
4. What is the first thing to check when a deployed app fails?

Answers: 1. In Secrets, read from the environment. 2. The form can be bypassed, so the server must check too. 3. Nothing. 4. The logs, and whether the same secret and run command exist there.

Part VII. Four app examples you can build

App	Users	Core feature	Data stored	Risk to watch
Feedback form	Fake customers	Form, validation, a thank-you page	Rating and comment (no names)	Abusive text; add a length limit
Quiz app	Learners	Questions, scoring, a results page	Score and time	Answer key exposed in the page source
Inventory tracker	Shop staff (test)	Add and update stock, low-stock alert	Item, count, updated time	Wrong counts; keep a change log
Dish Description Studio	Sellers (test)	AI description with a cap	Dish, price, description, token counts	Cost growth; cap output and log usage

VII.1 Build the quiz app (fake data, about 60 minutes)

1. **Plan:** five questions, one correct option each, a score out of five.
2. **Data:** store the questions in a list in code for this practice.
3. **Route:** show one question at a time; record each answer in the session.
4. **Score:** compare with the answer key on the server, never in the page.
5. **Results:** show the score and the questions answered wrongly.
6. **Pass when:** a full run gives the correct score, and reloading the page does not reveal the answer key.

VII.2 Build the inventory tracker (about 90 minutes)

1. **Table:** items with id, name, count, updated_at, and a change log table with item id, change, time.
2. **Routes:** list items, add an item, change a count. Every change writes a log row.
3. **Validation:** the count cannot go below zero; the name cannot be empty.
4. **Alert:** items at or below a threshold show a low-stock label.
5. **Pass when:** a change appears in the log, a negative count is rejected, and the low-stock label appears at the threshold.

Part VIII. Architecture in one page

Layer	What it does	Where it lives in Replit
Page	Shows the form and results	The web files in the project
Routes	Receive requests, validate, call the database or the model	The server file
Data	Stores records	The database option you chose
Secrets	Keys and private settings	The Secrets panel
Deployment	The public address and its run command	The Deploy panel

Keep the layers separate. The page never talks to the database directly, and the model key never reaches the page.

Part IX. Deployment and operations checklist

Item	Done
Secrets set in the deployment, not only in the workspace	[ ]
Run command matches what worked in the workspace	[ ]
Health route returns ok	[ ]
Logs checked after the first deployment and after each change	[ ]
Checkpoint saved before each deployment	[ ]
Usage and spending limit checked on the provider and Replit pages	[ ]
Test data removed from any production store	[ ]

Part X. Mobile, accessibility and polish

- Test the deployed address on a phone-width screen; forms must stack to one column.
- Every input has a visible label, and error messages sit next to the field.
- Keyboard users can reach every control in a sensible order.
- Colour is never the only signal for an error or a status.

Part XI. Exit exercise

1. Choose one app from Part VII and write its plan on one page.
2. Build the page, one route with validation, and one stored record, using fake data.
3. Deploy, then test a normal input, an empty input, a bad input and a missing secret.
4. **Pass when:** all four tests give the expected message, and the deployed app matches the workspace.

Part XII. Data, teams and change control

- **Schema first:** write the table list and the fields on one page before you create anything.
- **Migrations:** change a table in one small step, test it on fake data, then deploy; never edit a live table by hand.
- **Backups:** export the data on a schedule you set, and test one restore on a second project.
- **Teams:** one owner per project; reviewers read changes before they reach the deployed app.
- **Branches or checkpoints:** use a checkpoint before each risky change, and name it with the date and the reason.

XII.1 Exit test for the team workflow

1. Create a checkpoint, make a change, deploy to the test address, then roll back to the checkpoint.
2. **Pass when:** the rolled-back app matches the earlier behaviour and the logs show both steps.

Part XIII. Choosing Replit, Lovable or n8n for a job

Job	Best first choice	Why
A web page or app that users use directly	Replit or Lovable	Builds the interface and the server together
Moving data between apps on a schedule or event	n8n or Make	Connectors and routing without writing a server
A custom app that needs a workflow behind it	Replit for the app, n8n for the workflow	Each does the job it is designed for; connect them with a webhook

Check the current feature list for the tool you choose; products change quickly.

Part XIV. Frequently asked questions

Question	Answer
Do I need to install anything?	No. You work in the browser. Install only what your project's packages need, through the project itself.
Where do I keep an API key?	In the Secrets panel, read from the environment. Never in code, screenshots or chat.
Why does it work in the preview but fail when deployed?	Check the deployment's run command, the port setting, and whether the secrets exist in the deployment.
How do I undo a bad change?	Return to the last checkpoint in version history, then find the cause in the console.
Can I share my project?	Yes, with a test teammate and fake data only. Do not share secrets.
Should I use the Agent for everything?	Use it for drafts and small features. Read every change and test it yourself.
What about plans and prices?	Check Replit's live pricing page on the day you decide. This guide does not list prices.

Part XV. Common mistakes and fixes

Mistake	Fix
Hard-coded key in the code	Move it to Secrets and rotate the key at once
No server-side validation	Validate in the route, and return a clear message
No output cap on an AI call	Set max tokens and log the usage
Deploying without testing failure paths	Run the four failure tests in Part VI
Real personal data in a practice app	Replace with fake data and delete the real rows

Part XVI. Ten build recipes, from small to larger

Each recipe is one short build with a pass check. Do them in order; each one teaches a skill the next one needs.

#	Recipe	New skill	Pass when

1	A page that says hello, with your name	Run, preview, edit	Change is visible after refresh
2	A form that echoes the input back	Forms and routes	Submitted value appears on the next page
3	Validation with two rules	Server checks	Bad input returns a clear message
4	A counter stored in memory	State within one run	Count rises by one per click
5	A saved list in a database table	Storing records	Records survive a page reload
6	A secret read from the environment	Secrets	Code contains no key; the app still works
7	One AI call with a 50-word cap	Model calls and limits	Output stops near 50 words; usage is logged
8	A history page of saved AI results	Reading stored data	Newest first; no personal data shown
9	A checkpoint, a change, and a rollback	Version history	The rolled-back app matches the old behaviour
10	A deployed copy at a public address	Deployment and logs	Normal, empty, bad, and missing-secret tests pass

> **Stop rule for a recipe:** if a pass check fails twice, read the first red console line, change one thing, and test again. Do not change several things at once.

Part XVII. Worked walkthrough: Dish Description Studio, recipes 1 to 10

1. **Recipe 1–2:** the page shows the title "Dish Description Studio" and a form with dish, category and price. Submitting shows the same values on the next page.
2. **Recipe 3:** the server rejects an empty dish name and a price of zero, each with a clear message. Nothing is saved for a rejected input.
3. **Recipe 4:** a small counter shows how many descriptions this session has created, so you can see the route runs once per submission.
4. **Recipe 5:** each successful submission writes one row: dish, category, price, time. Reload the page to check the row survived.
5. **Recipe 6:** the model key lives only in Secrets. Remove it for one test to confirm the friendly "not configured" message, then put it back.

6. **Recipe 7:** the AI call uses a fixed system instruction, a 120-token output cap, and logs the token counts. Check that a 50-word description results.
7. **Recipe 8:** the history page lists the newest rows first. It shows dish, category, description and time, with no contact details.
8. **Recipe 9:** save a checkpoint, change the heading, then roll back. The heading returns and the rows remain.
9. **Recipe 10:** deploy, open the public address in a private window, and run the four tests from Part VI on the deployed copy.

Final check: the deployed app should behave the same as the workspace. If it does not, compare the run command, the secrets and the logs before changing any code.